ECE-6483 Real Time Embedded Systems
Homework 1

New York University, Fall 2024

Due on 02/18, 11:59 PM

Name: Raman Kumar Jha
NYU ID: N13866145

1: Using the variable x give definitions for the following:

(a) An integer: `int x;`

(b) A pointer to an integer: `int *x;`

(c) An array of 10 integers: `int x[10];`

(d) An array of 10 pointers to integers: `int *x[10];`

2. What is the output of the following C program?

```
#include <stdio.h>
int main ()
{
    int vals[5] = {4, 3, 2, 5, 1};
    int i;
    for (i=0; i<=5; i++) {
        printf("vals[%d]=%%\n", i, vals[i]);
    }
    return 0;
}
```

The program will output:

```
vals[0]=4
vals[1]=3
vals[2]=2
vals[3]=5
vals[4]=1
vals[5]=<undefined/garbage>
```

Accessing `vals[5]` can lead to undefined behavior, in this situation.

3. (a) What is the output of the following C program?

```
# include <stdio.h>
void fun(int y)
{
        y = 30;
}
int main()
{
    int y = 20;
    fun(y);
    printf("%d", y);
    return 0;
}
```

The output of the program is:

```
20
```

The variable `y` is passed by value, so changes inside `fun()` do not affect the original variable.

(b) In the program above, is the variable y in main() stored on the stack or on the heap?

The variable y in `main()` is stored on the **stack** because it is a local variable.

(c) What is the output of this C program?

```
# include <stdio.h>
void fun(int *y)
{
    *y = 30;
}
```

```
int main()
{
        int y = 20;
        fun(&y);
        printf("%d", y);
        return 0;
}
```

The output of the program is:

```
30
```

The function modifies the value at the memory location pointed to by `y`.

(d) In the program above, is the variable y in main() stored on the stack or on the heap?

The variable y in `main()` is stored on the **stack** because it is a local variable.

(e) True or false: &y in main() and y in fun() have the same value.

The statement &y in `main()` and y in `fun()` have the same value is **false**. In this case, $\&y$ in `main()` refers to the address of y, while y in `fun()` is a pointer that holds that address.